Throughput Characterization of a PC with a High Speed ATM Network Interface [*]

C.L. Chang, T.C. Hou, Y.S. Chu, K.J. Chen
National Chung Cheng University
Department of Electrical Engineering
Chia-Yi, Taiwan 621 ROC
tch@ee.ccu.edu.tw

Abstract

With the broad acceptance of the ATM technology, development of the related host-interface has been actively carried out. We have built an ATM/TAXI network interface card for the ISA-bus based Personal Computer (PC) host. Two FPGAs that support the TAXI interface and the ATM/AAL5 functions are implemented for such a high speed communication interface. Based on this prototype, we measure individual communication protocol stack throughputs and compare them with those of the Ethernet network. We then derive a design guideline for improving the system throughput of an ATM PC host.

*This work is supported in part by the ROC National Science Council research grant (NSC-85-2213-E-194-022) and in part by ITRI Computer & Communication Research Laboratories (G4-85012-12) under the sponsorship of the Ministry of Economic Affairs, ROC.

1 Introduction

Recently, we have seen the demand for multimedia communications grows at a fast speed. As this demand requires higher bandwidth and better Quality of Service (QoS) than what the conventional LAN (e.g., Ethernet) can support, alternative networking approaches are being sought to satisfy this demand. ATM technology is generally considered the most promising networking technology to provide transport service to applications with widely different traffic characteristics. ATM can offer transport. speed at higher than 100 *Mbps* and its connection-oriented approach makes it easier to provide QoS guarantee to the users. Research and development efforts in this area have been actively carried out in the past few years.

We have designed ATM/AAL5 chips and built a high speed ATM/TAXI network interface card (NIC) for use in the ISA-bus based PC host. Our intent was to come up with a prototype subsystem that allows us to further investigate the end-to-end performance impacts, when the multimedia application is transported. However, upgrading the networking interface to a high speed ATM interface does not automatically increase the end-to-end application throughput. It is well known that the higher layer networking protocols, e.g., the transport layer TCP protocol, are the bottleneck of the end-to-end transmission path over a high speed communication network.

Many studies [1, 2, 4, 7, 9] have been conducted that discovered the cost of the transport protocol overhead. The consensus is that the memory copy operation in the implementation of the TCP protocol is the main reason that TCP consumes much CPU time [4]. Within the TCP protocol, the memory access for checksum calculation also consumes a significant portion of the protocol processing time [2, 7]. Besides the TCP protocol processing, other operations are also required for moving the data from the application to the network. These operations, e.g., the driver for the networking device, also consume CPU time.

In this paper, we concentrate on the throughput characterization of the NIC and the device driver. For both the DOS and the Windows PC communication environments, we measure the CPU overhead of each function block on the end-to-end communication path and compare the end-to-end throughputs between the Ethernet network and ATM network platform. From these studies, we know quantitatively the contribution from each part of the communication path to the CPU overhead. We also derive a design guideline for improving the end-to-end communication throughput of a PC-ATM platform, and a formula to predict the system throughput for different 80x86 CPU PC platform.

The rest of this paper is organized as follows. In section 2, we present the ATM NIC architecture that uses a hardware approach to process the ATM/AAL5 protocol. The pipeline operation is adopted to eliminate the store-and-forward data transmission latency in the NIC. We describe the testing environment for the ATM NIC in section 3. In section 4, we show the measurement results for each communication components in different PC configuration platforms and derive a design guideline for improving the system performance. Section 5 summarizes our studies.

2 Hardware Architecture

The ATM host protocol architecture can be viewed as a stack of layers which are illustrated in Fig. 1 and the PC system architecture is shown in Fig. 2. When we began to design the host interface card, we wanted it to support all current Internet applications that ran on the Ethernet PC DOS/Windows 3.1 environment. From this viewpoint, the easiest method is to replace the Ethernet packet driver by the ATM device driver. We develop our ATM device driver which is compatible with the Ethernet packet driver and provides all interface specified in the FTP packet driver specification [6] to the upper layer. In Fig. 3, we show the high-level ATM network interface card hardware architecture for the ISA-bus PC. To leave the host CPU more time to process upper layer protocol, we design two FPGA ASICs (ATM_Tx and ATM_Rx) which perform all cell-based functions in the transmitter and the receiver. The ATM_Tx uses the internal 8-bit bus design, and is clocked at 12.5 MHz. The design is a 3-stage pipeline structure to achieve high performance and minimum buffering of data. The ATM_Tx FPGA's main features are: In the AAL layer, we implement the ITU-T I.363 AAL5 recommendation [8], such as non-assured data transfer, pipeline operation, 0-47 octets padding, CRC-32 generation [3, 5], construction of the CS-PDU (Convergence Sublayer Protocol Data Unit) from the CS-SDU (Convergence Sublayer Service Data Unit), and SAR sublayer functions. In the ATM layer, we provide the generation of the 8-bit HEC in the ATM header, "more-bit" (ATM-user-to-ATM-user indication bit of the PT field) setting for the AAL5 protocol, and the 53-octet ATM cell generation. To reduce the ATM NIC cost and area, we also include the TAXI_Tx controller in the ATM_Tx FPGA. The above functions are implemented by hardware without any software's intervention. The design philosophy for the ATM_Rx follows a similar idea as the ATM_Tx to realize the ATM and AAL5 layer receiving functions.

2.1 Data and Control Flow for the Transmitter

The major functions for the transmission part of the host interface are:

1. move the CS-SDU from the host memory to the Tx_Buffer in the NIC,
2. convert the CS-SDU to a CS-PDU,
3. fragment the CS-PDU into a sequence of 48-octet ATM payloads,
4. prefix the 5-octet ATM header to the 48-octet ATM payload to form an ATM cell, and
5. feed ATM cells to the TAXI link.

In the Windows 3.1 environment, when the application has data to be transmitted, it calls the Winsock entity to generate a CS-SDU. When the Winsock has a CS-SDU to send, it requests the ATM device driver to transmit this CS-SDU over the ATM network to the peer entity. When the ATM device driver receives the upper layer request to send a packet, the first step is to detect whether the ATM_Tx chip is in the free state. If the ATM_Tx is in the free state, the CS-SDU is allowed to be transmitted over the ISA bus to the interface card. Next, the device driver sets the ATM_Tx chip's registers (packet length, VCI/VPI value, and start flag of the ATM_Tx). After the ATM_Tx has been notified, the device driver moves the CS-SDU to the Tx_Buffer. The operation at the ATM_Tx uses a pipeline technique. In other words, moving data from the host to the Tx_Buffer and the processing of the ATM/AAL5 function at the ATM_Tx are done concurrently. Because the ATM/AAL5 operation rate is 100 Mbps, which is much faster than the ISA bus speed, the Tx_Buffer has a probability of being empty. When the Tx_Buffer is empty, the ATM_Tx needs to halt until further data arrives in the Tx_Buffer.

The TAXI_Tx controller monitors the TAXI_Tx FIFO status and decides whether it needs to request the TAXI to transmit an ATM cell or not. It also provides the TAXI chip the required signals during transmission to satisfy the ATM forum UNI physical layer specification.

2.2 Data and Control Flow for the Receiver

When the message is transferred across the ATM network, it has been segmented into a sequence of cells. In ATM, the recurrence of cells containing information from an individual user is not necessarily periodic. Moreover, due to the statistical multiplexing effect, cells from different messages that are received at the receiver may also arrive with arbitrary interleaving. Thus, the receiver has to concurrently reassemble these interleaved cells to each individual message and then forward messages to the upper layer [8].

In our implementation, we use a semi-dynamic bank allocation method for the reassembly buffer design. The Rx_Buffer memory is managed by the CAM (Content addressable Memory) which is a fully customized design within the ATM_Rx chip. The receiver also performs the following functions:

1. delineates the ATM cell out of the TAXI chip (done by the TAXI_Rx controller),

2. processes the ATM cell which is temporarily stored at the TAXI_Rx FIFO, including the VCI/VPI, HEC, and the "more-bit" for AAL5 in the 5-octet

ATM header,

3. extracts the SAR-PDU from the ATM cell,
4. identifies the corresponding receiving bank number for the received cell for re-assembling CS-PDU,
5. performs the CS-PDU's CRC-32 calculation,
6. interrupts the host as soon as the CS-SDU is completely reassembled.

At the receiver side, if the TAXI_Rx controller determines that the TAXI_Rx FIFO has at least one cell, it signals the ATM_Rx to receive the ATM cell from the TAXI_Rx FIFO. When the ATM_Rx receives the request from the TAXI_Rx controller, it begins the reassembly functions for this received cell. First, the ATM_Rx calculates the HEC of the ATM cell header. If the HEC is in error, then it discards this received cell. Otherwise, it searches the internal CAM which records the reassembling connection information, such as the temporary CRC-32 value, the packet length, VCI/VPI, and the bank status (*free, busy*, or *completed-reassembly*). If the incoming cell belongs to a new connection, then it searches a free bank from CAM for reassembling this cell. If the incoming cell is the last cell (more-bit=1), the reassembly of the CS-PDU is completed.

When the ATM_Rx has a reassembled CS-PDU, it generates an interrupt signal to the host. When the host receives an interrupt signal, it then executes the interrupt routine (the *receiver()* function of the device driver). When the *receiver()* of the device driver is started, the first step is reading the CS-SDU's information from the ATM_Rx. The information includes the length of the CS-SDU, the bank number, and the CRC result (error or correct). The second step is moving the CS-SDU from the Rx_Buffer to the host memory. When the CS-SDU has been completely moved to the host, the third step is releasing the reassembly bank. This bank is then available for the ATM_Rx to reassemble other CS-SDUs.

3 Testing Environment

Two ATM NICs are used for point to point test. In the first step of the testing, we focus on testing the ATM_Tx and ATM_Rx FPGAs. The test includes interface testing and ATM/AAL5 protocol operation testing. In the ATM/AAL5 protocol operation testing, besides the HEC, CRC-32, and more-bit tests, different CS-SDU packet lengths are used to test the padding function (no padding, padding at two cells, and one cell padding) of the CS-PDU.

After the point-to-point tests are verified, we also use the HP ATM protocol analyzer to test the ATM NIC. In testing the transmitter, the ATM protocol analyzer can correctly receive CS-SDU from our ATM NIC. In testing the receiver, our ATM NIC can also receive the CS-SDU from the ATM protocol analyzer. Through the ATM protocol analyzer verification, we ensure that our ATM/AAL5 protocol operation is correct. Finally, we combine the hardware and software to do the system testing. The testing environment is shown in Fig. 4.

We use Fore Systems' ForeRunner ASX-200 ATM LAN switch as our ATM switch. As Fig. 4 shows, we have successfully demonstrated the following applications: In the DOS environment, we can execute FTP, Telnet, and several network games via the ATM NIC over the Fore switch. In the Windows environment, we build a WWW server in one PC, and run the NETSCAPE in another PC to access WWW server information. Furthermore, we use the Video Phone On Network (VPON) product from the Formosa Industrial Computing, Inc. to demonstrate the video phone application through our ATM NIC cards. The VPON uses the motion JPEG compression and runs on the Windows 3.1 operating system. The above tests are all working correctly.

4 System Throughput Evaluation

Once we have successfully built a high speed ATM NIC, the next step is to understand how the application throughput is improved. It is well known that the end-to-end throughput at the application level does not scale up accordingly as the NIC speed is increased. We thus would like to know in what way the end-to-end system throughput is affected by the various components on the end-to-end path. In this section, we report the measurement results on different PC platforms. We conduct a series of measurements to find out the CPU overhead on each part of the communication path within the PC. Specifically, the system performance improvement is quantified, when the 10 Mbps Ethernet network interface is upgraded to the 100 Mbps ATM network interface and when the CPU is upgraded to a more powerful one. From these measurements, we derive a formula that predicts the system throughput for different 80X86 CPU PC platforms. With all these knowledge, we are able to point out which part of the communication path needs to be improved, if we want to fully utilize the ATM link bandwidth.

4.1 Measurement Configuration

In general, the PC communication environment can be divided into two categories. One is the DOS environment and the other is the Windows environment. In the DOS environment, all networking applications (AP) are running on top of the packet driver. There are three main function blocks in moving the data between the AP and the network: the NIC, the packet driver, and the DOS AP. There is only one data copy operation from the AP to the NIC or from the NIC to the AP in the DOS communication environment. With the ISA bus, moving data between the host and the NIC is done asynchronously by the host CPU (it will consume some CPU time). When the NIC receives a packet from the network, the interrupt method is used by most Ethernet NICs to ask the host CPU to copy the packet from the NIC to the host memory. Note that, the Ethernet NIC operates in a store-and-forward manner on the transmission side. The extra Ethernet transmission latency may have an impact on whether the transmission side or the receiving side is the bottleneck of the transmission path.

In the Windows 3.1 environment, most networking APs are running on top of the Winsock interface. Its function blocks consists of the NIC, the packet driver, Winpkt, Winsock (TCP/IP), and the Windows AP. In this communication flow, there are three data copy operations in addition to the TCP internal memory access for the checksum calculation [4]. These three copy operations are done between the NIC and Winpkt, between Winpkt and Winsock, and between Winsock and AP.

The measurement environment is shown in Fig. 5 (the shadow areas are where our test programs are). We use two PCs interconnected by an Ethernet and an ATM network, respectively. The PC with the 80486-33 CPU is used as the reference platform. While the measurement is done over an Ethernet, we do not allow other stations to send data on the Ethernet. Before we do the packet driver measurement, we use the Ethernet protocol analyzer (Sniffer) to obtain a few key parameters for the DOS FTP transaction. We discover that the maximum Ethernet frame length is 566-byte. Since the combined overhead of TCP, IP, and Ethernet is 54-byte, we know that the TCP SDU length is 512-byte.

We first write a test program on top of the packet driver in the DOS environment to measure the processing overhead (in ms/Kbyte) due to either the Ethernet driver or the ATM driver. As Fig. 5 shows, one PC station at the transmitting side sends data to the packet driver, which allows us to measure the transmitting driver overhead, and the other PC station receives the data from the packet driver, which allows us to measure the receiving driver overhead. Every transaction contains 10-Mbyte data, which is sent as 2048 512-byte TCP SDUs. A logic analyzer is used to measure the CPU cost in moving the data between the host and the NIC over the ISA bus.

In the Windows environment, we write a client/server test program (the client sends data to the server) over the Winsock interface. On each PC, 4096 bytes of buffer is pre-allocated for the Winsock. In the client, the Winsock's maximum transmission unit (MTU) is 660 bytes. We send 1 Mbyte data in the Winsock test program from the client to the server to measure the combined overhead due to both the Winsock and the driver. The Winsock overhead can be derived by deducting the driver overhead from the combined Winsock/driver overhead.

4.2 Measurement Results

Table 1 shows our measurement results for the DOS environment. Note first the NIC latency difference betweeen the Ethernet NIC and the ATM NIC. In general, the Ethernet NICs use a store-and-forward approach in moving data from the host to the network. Each 1000 bytes incur $0.8msec$ store-and-forward delay on a $10Mbps$ Ethernet NIC. For the ATM NIC, a pipeline approach is taken. Hence, there is only a negligible NIC latency. This NIC latency has an impact on the end-to-end throughput in certain circumstances as will be discussed later. Table 1 also lists the throughput and the CPU cost for the transmitter driver, the receiver driver (including interrupt processing), moving the data across the bus, and the AP limit. The values in the "AP limit" row of Table 1 represents the maximum AP throughput and the minimum CPU cost, respectively, provided by the protocols underneath the AP. In other words, the CPU cost in the AP limit is the sum of all CPU costs that are required to interface an AP with the NIC. In our case, it is equal to the sum of the receiver driver cost and the bus data copy cost. The last row of Table 1 shows the measured end-to-end AP (*ftp*) throughput. Regarding the driver costs at either the transmitter side or the receiver side, our ATM driver has a lower CPU cost than the Ethernet driver. The CPU cost for the bus data copy depends on the CPU and internal bus used. It is independent of whether an Ethernet NIC or an ATM NIC is used. As for the driver itself, the transmitting side costs less CPU time than the receiving side. This is because the Rx_driver complexity is typically higher than that of the Tx_driver and every reception of a packet introduces an interrupt overhead, which we include in the measurement of the Rx_driver. The receiving side is thus generally the slower part of the communication path in the DOS communication environment. However, the above statement is not always true in the Ethernet enrironment. As Figure 6 shows, on the transmitting side, there is a store-and-forward delay on the Ethernet NIC. This delay time is proportional to the packet length. If the sum of the Tx_driver overhead plus the Ethernet transmission latency is greater than the sum of the Rx_driver overhead plus the interrupt overhead, then the transmitting side becomes the slower part on the end-to-end transmission path. For our ATM NIC, we adopt the pipeline operation to eliminate this transmission latency. In this case, the receiving side is the bottleneck.

	10 Mbps Ethernet ISA Bus		100 Mbps ATM ISA Bus	
NIC latency	0.8ms/Kbyte		negligible	
	throughput	CPU cost	throughput	CPU cost
Tx driver	850Kbyte/s	1.18ms/Kbyte	5076Kbyte/s	0.19ms/Kbyte
Rx driver and int	438Kbyte/s	2.28ms/Kbyte	800Kbyte/s	1.25ms/Kbyte
Bus data copy	950Kbyte/s	1.05ms/Kbyte	950Kbyte/s	1.05ms/Kbyte
AP limit	300Kbyte/s	3.33ms/Kbyte	435Kbyte/s	2.30ms/Kbyte
ftp ETE	137Kbyte/s	7.30ms/Kbyte	210Kbyte/s	4.76ms/Kbyte

Table 1. Measurement results in the DOS environment.

Let T_{xmit}, T_{rcvr}, T_{bus}, T_{ftp} be the CPU overheads due to executing the Tx_driver codes, executing the Rx_driver codes (except moving data across the ISA bus), moving data across the ISA bus, and executing the FTP codes, respectively. Let T_{ete} and R_{ete} be the CPU overhead and the throughput of the end-to-end FTP transmission. Also denote T_{nic} as the store-and-forward delay incurred in the network interface card. If the receiver is the slower part of the communication path, we have

$$T_{ete} = T_{rcvr} + T_{bus} + T_{ftp} . \tag{1}$$

If the transmitter is the slower part of the communication path, we have

$$T_{ete} = T_{nic} + T_{xmit} + T_{bus} + T_{ftp} . \tag{2}$$

In either case, if the NIC throughput is higher than $\frac{1}{T_{ete}}$, then

$$R_{ete} = \frac{1}{T_{ete}} .$$

In both the Ethernet and the ATM configurations of our measurements, the receiving side is the slower part of the transmission path. Hence, we expect equation 1 to hold. Indeed, for the $100Mbps$ ATM NIC, we have $4.76 = 1.25 + 1.05 + 2.46$. However, for the $10Mbps$ Ethernet NIC, we obtain $7.30 > 2.28 + 1.05 + 2.46$. To resolve this discrepancy, we resort to an Ethernet protocol analyzer which is used to capture all Ethernet frames for this FTP transaction. To our surprise, there are many ACK frames which we did not account for previously. It reveals that the DOS FTP protocol is a full duplex protocol, which is not suitable to run on the Ethernet, as the ACK frames flow in the opposite direction of the FTP data frames, causing collisions on the Ethernet. Among the $7.30ms/Kbyte$ overhead of the DOS FTP transaction, the collision overhead is $1.54ms/Kbyte$, quite a significant figure. In an ATM network, which is a full duplex environment, there is no collision problem.

Table 2 shows our measurement results for the Windows 3.1 environment. The first row combines the Rx_driver and bus data copy values in Table 1. The entries in the AP limit are obtained from running the Winsock test program. We also measure the actual memory copy time for the Intel 80486 CPU. By deducting the receiver driver CPU cost and twice the memory copy cost from the CPU cost in the AP limit, we obtain the CPU cost of the Winsock and Winpkt. In both the Ethernet and the ATM cases, the Winsock CPU costs come out at $3.95msec/Kbyte$, which in a sense validates our measured results. The entries in the last row of Table 2 are obtained from running the Windows *ftp* program. When deducting the CPU cost of the AP limit from the CPU cost of the end-to-end *ftp*, we get the CPU cost of the *ftp* code itself. The values are $2.32msec/Kbyte$ and $2.28msec/Kbyte$ for the Ethernet and the ATM case, respectively. The difference is less than two percent. The collision problem on the Ethernet Windows 3.1 environment does not further reduce the the end-to-end throughput.

Table 2 also shows that, in the Windows environment, upgrading the Ethernet NIC to a high speed ATM NIC does not increase the end-to-end AP throughput by much. In the PC Windows environment, most of the processing overheads come from the Winsock+Winpkt portion of the protocol stack. Among the Winsock+Winpkt protocol, a large portion of the overhead come from the TCP/IP protocol processing. The two data copy operations is roughly 10 percent of the protocol processing. Figure 7 shows the percentages of each processing overheads of the 486-33 ISA-bus host with either the Ethernet or the ATM NIC.

	10 Mbps Ethernet ISA Bus		100 Mbps ATM ISA Bus	
	throughput	CPU cost	throughput	CPU cost
Rx_driver + bus	300Kbyte/s	3.33ms/Kbyte	434Kbyte/s	2.30ms/Kbyte
Winsock+Winpkt	253Kbyte/s	3.95ms/Kbyte	253Kbyte/s	3.95ms/Kbyte
memory copy x 2		0.40ms/Kbyte		0.40ms/Kbyte
AP limit	130Kbyte/s	7.68ms/Kbyte	150Kbyte/s	6.65ms/Kbyte
ftp ETE	100Kbyte/s	10.0ms/Kbyte	112Kbyte/s	8.93ms/Kbyte

Table 2. Measurement results in the Windows 3.1 environment.

We next want to understand how much system throughput can be gained when the ISA bus is upgraded to the PCI bus, and when the 486-33 PC platform is upgraded to the 586-100 PC platform. The throughputs for these eight different configurations (ISA bus vs. PCI bus, Ethernet NIC vs. ATM NIC, and 486-33 PC vs. 586-100 PC) are shown in Tables 3 and 4. Note that most PC Ethernet interface cards are designed for the ISA bus. The CPU is responsible for moving data between the host and the network interface. For the PCI interface, moving data between the host and the NIC can be done by the DMA function which doesn't cost any CPU time. Therefore the high speed DMA function of the PCI bus does not contribute negatively to the end-to-end throughput.

In the 486-33 PCI platform, the receiver is the slower part of the communication path, therefore the end-to-end AP throughput is upper-bounded by

$$R_{ete} < \frac{1}{T_{rcvr} + T_{winsock}}. \tag{3}$$

When a high speed PCI bus is used, the bus data movement overhead is negligible, resulting in approximately 16 percent improvement in the system throughput. This is because the CPU-time-consuming upper layer (Winsock) protocol is the bottleneck of the system throughput. The overhead percentages of each function blocks based on the 486-33 PCI bus platform are shown in Figure 8. Note the increase in each function block's percentage when compared with those in Figure 7.

486-33 PC Platform				
	Ethernet ISA Bus	ATM ISA Bus	Ethernet PCI bus	ATM PCI bus
Bus data copy	950 Kbyte/s		high	
driver throughput	R_x 300 Kbyte/s	R_x 450 Kbyte/s	R_x 445 Kbyte/s	R_x 800 Kbyte/s
2 times memory copy	2532 Kbyte/s			
winsock winpkt protocol	253 Kbyte/s			
Max AP	130 Kbyte/s	150 Kbyte/s	150 Kbyte/s	178 Kbyte/s

Table 3. Throughput Comparisons of four different configurations with 486-33 CPU.

As the CPU is upgraded to 586-100, the CPU time to process Winsock, Winpkt, driver, and interrupt is reduced. For configurations with the Ethernet NIC, the Tx_driver overhead plus the Ethernet transmission latency (with the packet length equal to 512 bytes) is greater than the sum of the Rx_driver and the interrupt overheads. In this case, the transmitting side is the slower part of the driver. The corresponding processing overhead percentages are shown in Figure 9. Figure 9 shows that, for the ISA-bus case, the NIC store-and-forward latency (26%) and moving data across the bus (29%) each consumes comparable percentage of CPU time as the Winsock/Winpkt (32%). When the ISA bus is replaced by the PCI bus, the NIC store-and-forward latency (37%) and the Winsock/Winpkt processing (45%) are the dominant components of the CPU overhead. This clearly indicates how large an impact the NIC store-and-forward latency can have on the end-to-end throughput. For configurations with the ATM NIC, there is no NIC store-and-forward latency. In this case, the receiving side is the slower part of the driver. The corresponding processing overhead percentages are shown in Figure 10. In the case of the ISA bus, the ISA bus overhead (40%) is comparable with the Winsock/Winpkt (43%) overhead. When the ISA bus is replaced by the PCI bus, the Winsock/Winpkt processing (72%) is the sole dominant component of the CPU overhead. This is the case (586-100 CPU, PCI bus, ATM NIC) which we can expect the highest AP throughput (less than 730 $Kbyte/s$) among all configurations we discussed. The efficiency of the Winsock/Winpkt implementation significantly affects the achievable AP throughput.

586-100 PC Platform				
	Ethernet ISA Bus	ATM ISA Bus	Ethernet PCI bus	ATM PCI bus
Bus data copy	1111 Kbyte/s		high	
driver throughput	T_x 500 Kbyte/s	R_x 833 Kbyte/s	T_x 910 Kbyte/s	R_x 3333.3 Kbyte/s
2 times memory copy	11764 Kbyte/s			
winsock winpkt protocol	1016 Kbyte/s			
Max AP	328 Kbyte/s	440 Kbyte/s	465 Kbyte/s	730 Kbyte/s

Table 4. Throughput Comparisons of four different configurations with 586-100 CPU.

As for the end-to-end throughput, Table 4 shows that, with a more powerful CPU, the configuration with the ATM NIC does show a higher percentage of increase in throughput than that with the Ethernet NIC. This is mainly due to the pipeline design of the ATM NIC.

From many measurements, we conjecture that the end-to-end throughput of a Windows 3.1 AP running on a 80x86-like PC with the PCI-bus, ATM NIC is bounded by

$$Max\ AP\ throughput\ <\ 0.8 * \frac{CPU\ \ speed}{40} * 178\ , \qquad (4)$$

where the CPU speed is measured by the benchmark tool of ZIFF-DAVIS' PC bench 9.0 version. To derive this throughput bound, we assume that the overhead due to moving data across the bus is negligible and the NIC adopts pipeline operations to eliminate the NIC store-and-forward latency. In this case, the majority of factors that affect the throughput are software overheads (Winsock, Winpkt, driver, interrupt, and memory-to-memory data copy) and they are fully dependent on the CPU computing power. Our empirical data shows that, if the CPU computing power increases by X times then the system throughput increase approximately by 0.8*X times (because different peripheral effects such as the cache size and the memory speed need to be factored into considerations). In this throughput conjecture, we use the 486-33 CPU measurement results as the reference value.

Based on the above results, we conclude that in order to fully utilize the high speed communication bandwidth provided by ATM and the PCI bus, The following considerations are critical.

1. In the NIC, we need to implement the pipeline design on the transmitter part to eliminate the NIC store-and-forward latency.

2. To reduce the bus data movement overhead between the NIC and the host, the PCI bus burst mode data movement (DMA) is preferred.

3. At the receiving side of the driver, the hardware interrupt that triggers the packet receiving function at the host also contributes significant CPU overhead. If the NIC implements the PCI bus master function, it is not necessary to use the hardware interrupt method to inform the host to do the data movement.

4. Data copy and Winsock protocol processing overhead is the dominant factor of the system throughput. Reducing the number of memory accesses in the implementation of the TCP protocol has a major impact on the system throughput.

5 Summary

An interface architecture based on our own designed ATM/AAL chipsets plus the TAXI chip for the transmission unit is developed. For supporting the ATM layer and AAL5 layer functions, we have implemented ATM_Tx and ATM_Rx FPGAs which conform to the specification recommended by the ITU-T. The transfer rate of the ATM network interface card is 100 Mbps. The interface architecture can concurrently perform all the related functions in parallel, which allows for further upgrade in the transmission rate from 100 Mbps to the OC-3 or higher rates.

With this ATM NIC, we measure the overhead due to each function block in the PC communication environment. System throughputs are measured/quantified when the underlying configuration is an ISA bus Ethernet, an ISA bus ATM, a PCI bus Ethernet, and a PCI bus ATM, with either the 486-33 or the 586-100 PC platform. From these measurements, we quantify the throughput overhead for the end-to-end communication. These communication overheads include: the NIC store-and-forward latency, moving data across the bus, the hardware interrupt at the receiving end, and the upper layer data copy and protocol processing. To fully utilize the network link bandwidth, every part of the communication path needs to be improved. The pipeline architecture can be used to eliminate the NIC store-and-forward latency. The PCI bus master function can be adopted in the NIC to eliminate the receiving hardware interrupt overhead and improve the performance for moving data across the bus. Transport layer protocol processing can be made faster by reducing the number of memory accesses. If all the above overheads can be properly reduced, the end-to-end system performance can be proportionly improved over a high bandwidth ATM link.

References

[1] W. Almesberger. High-Speed ATM Networking on Low-End Computer Systems. *http://lrcwww.epfl.ch/linux-atm*, August 1995.

[2] D. Bands and M. Prudence. A High-Performance Network Architecture for a PA-RISC Workstation. *IEEE J. on Selected Areas in Commun.*, pages 191–202, 1993.

[3] K. J. Chen and C. L. Chang. Universal Parallel CRC Circuit and Algorithm. filed for patent.

[4] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An Analysis of TCP Processing Overhead. *IEEE Commun. Magazine*, pages 23–29, June 1989.

[5] S. Dravida. Error Control Aspects of High Speed Networks. AT&T Bell Laboratories.

[6] FTP Software, Inc. *PC/TCP version 1.09 Packet Driver Specification.*

[7] J. H. Huang and C. W. Chen. On Performance Measurements of TCP/IP and its Device Driver. In *Proc. 17th Conference on Local Computer Networks*, pages 568–575, Sept. 1992.

[8] ITU-T. *B-ISDN ATM Adaptation Layer (AAL) Specification.*

[9] K. Moldeklev, E. Klovning, and O. Kure. TCP/IP Behavior in a High-Speed Local ATM Network Environment. In *Proc. 19th Conference on Local Computer Networks*, pages 176–185, Oct. 1994.

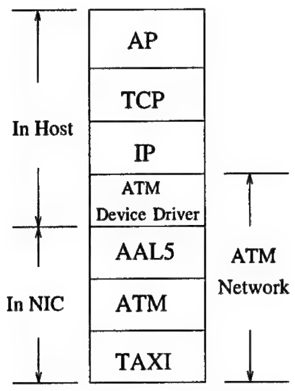


Figure 1: ATM host protocol architecture.

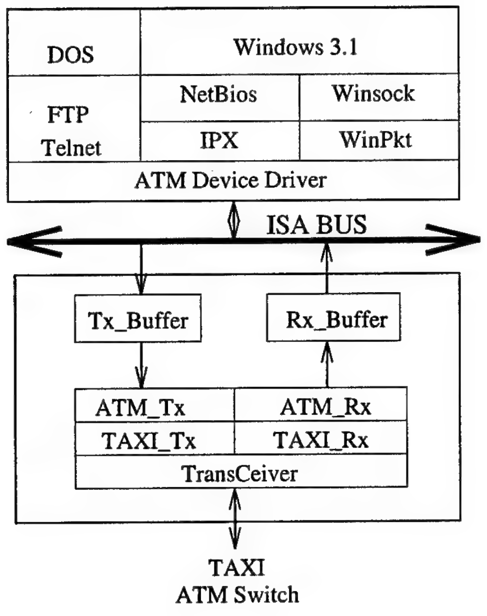


Figure 2: ATM host system architecture.

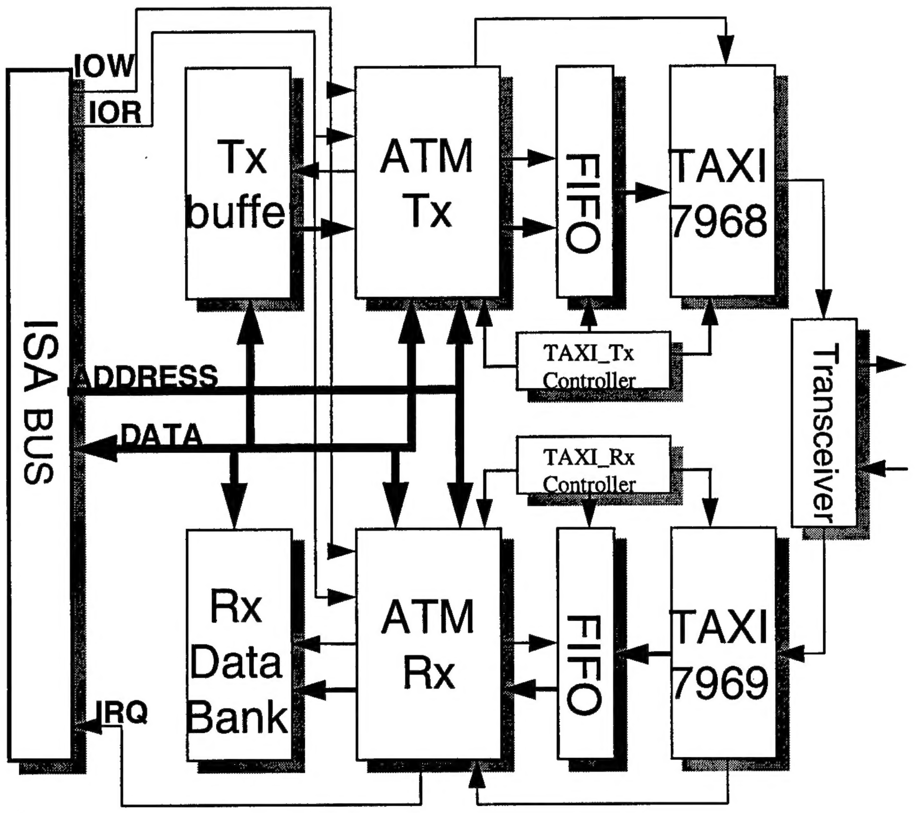


Figure 3: High-level ATM NIC hardware architecture.

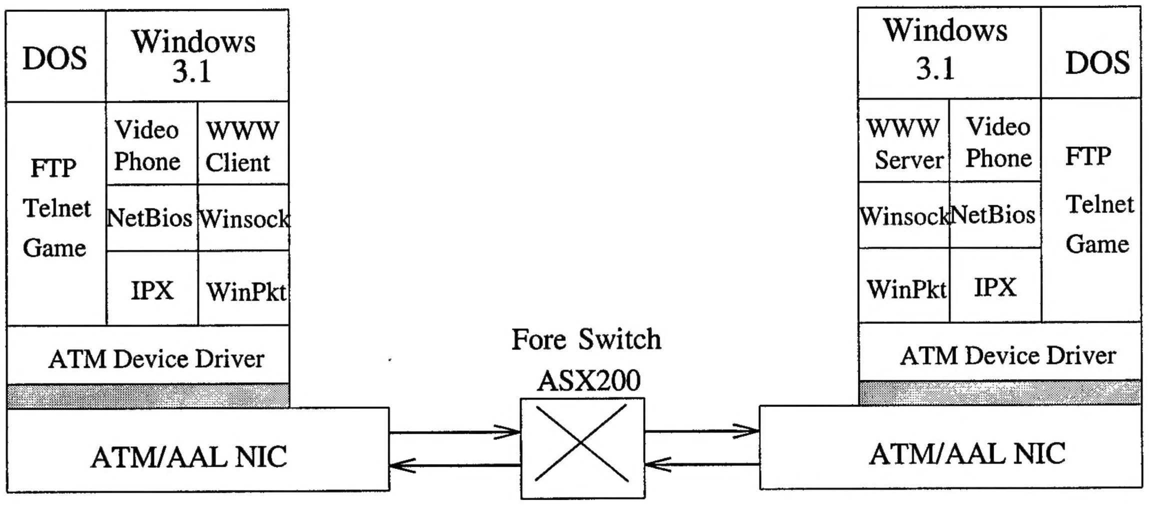


Figure 4: ATM NIC testing environment.

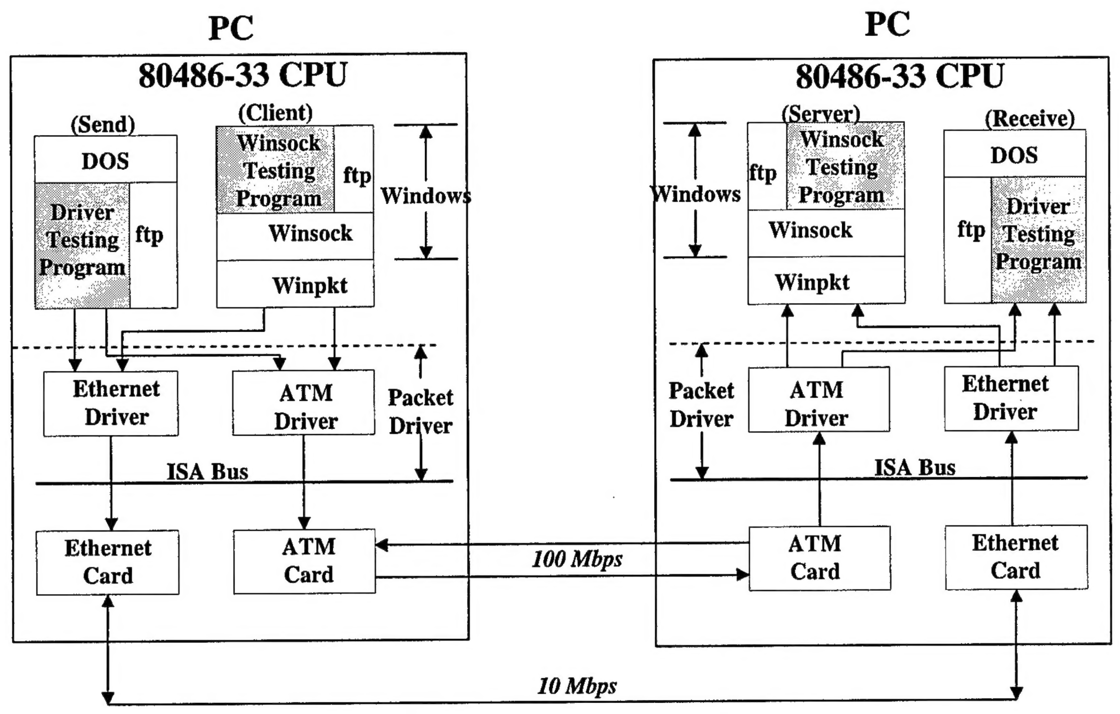


Figure 5: The measurement environment.

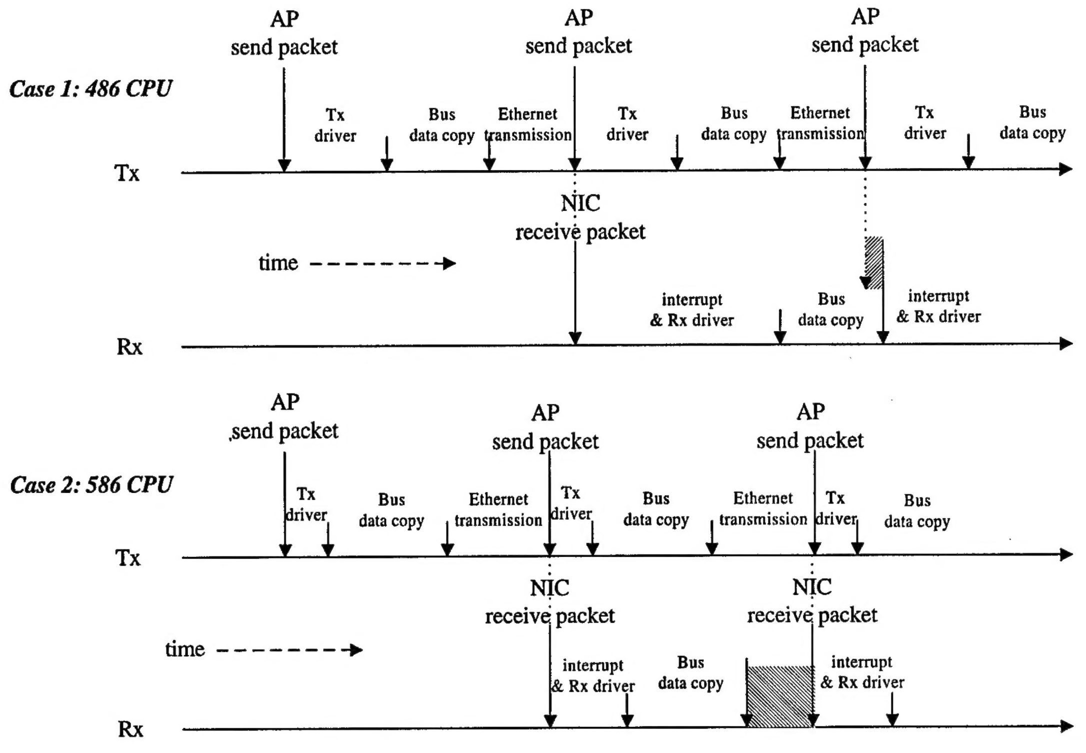


Figure 6: Packet transmission timing diagram in the DOS communication environment.

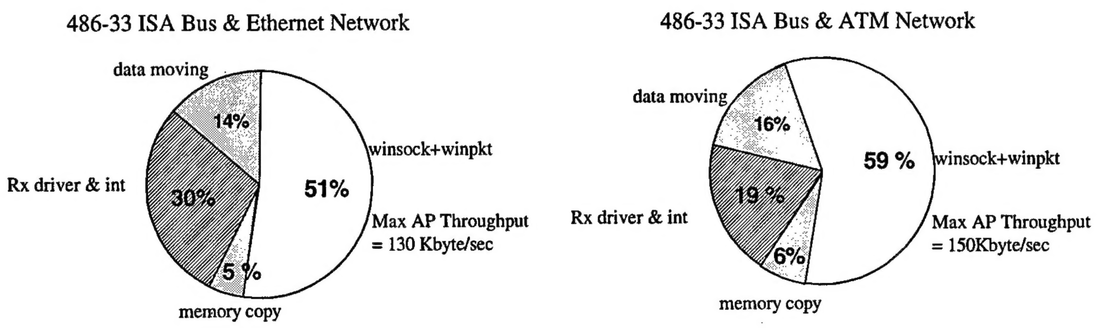
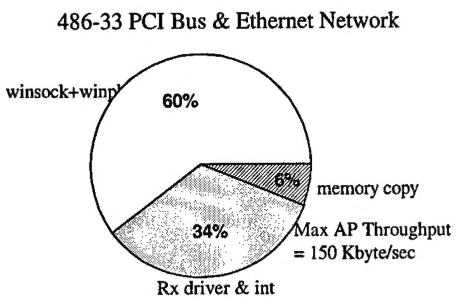


Figure 7: CPU overheads on the 486-33 ISA-bus platform.



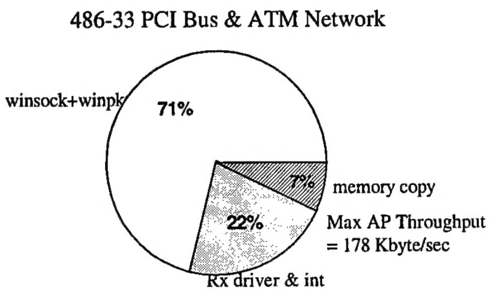
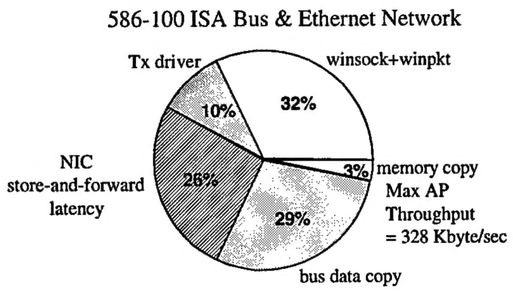


Figure 8: CPU overheads on the 486-33 PCI-bus platform.



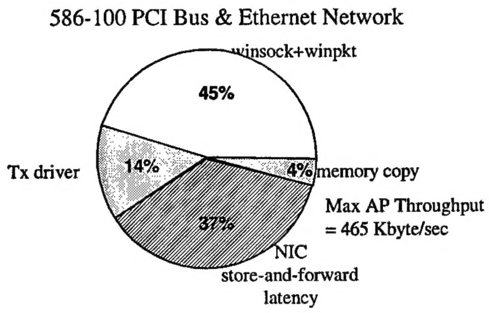
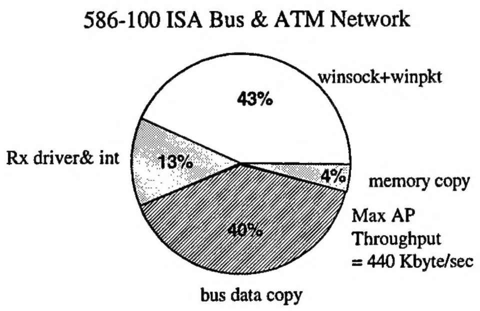


Figure 9: CPU overheads on the 586-100 Ethernet NIC platform.



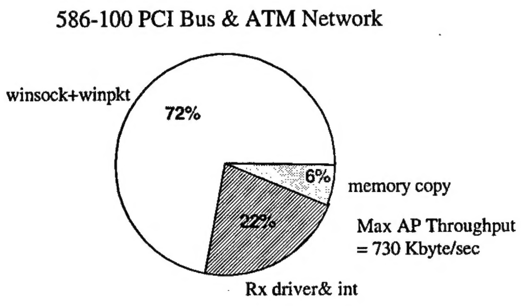


Figure 10: CPU overheads on the 586-100 ATM NIC platform.